Question Bank- OOPS

(BCS10T1006)

Q1. Differentiate between data abstraction and data encapsulation.

● Encapsulation Encapsulation is the process of combining data and functions into a single unit called class. In Encapsulation, the data is not accessed directly; it is accessed through the functions present inside the class. In simpler words, attributes of the class are kept private and public getter and setter methods are provided to manipulate these attributes. Thus, encapsulation makes the concept of data hiding possible.(Data hiding: a language feature to restrict access to members of an object, reducing the negative effect due to dependencies. e.g. "protected", "private" feature in C++).

● Abstraction We try to obtain an abstract view, model or structure of a real life problem, and reduce its unnecessary details. With definition of properties of problems, including the data which are affected and the operations which are identified, the model abstracted from problems can be a standard solution to this type of problems. It is an efficient way since there are nebulous real-life problems that have similar properties.

Q2. Discuss the use of public, private and protected access specifiers and their visibility in the class.

● Access Specifiers : The access specifiers are used to define how functions and variables can be accessed outside the class. There are three types of access specifiers:

1. Private: Functions and variables declared as private can be accessed only within the same class, and they cannot be accessed outside the class they are declared.

2. Public: Functions and variables declared under public can be accessed from anywhere.

3. Protected: Functions and variables declared as protected cannot be accessed outside the class except a child class. This specifier is generally used in inheritance.

Q3. Discuss default constructor and parameterized constructor with the help of an example in C++.

● Constructor : Constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally. The constructor in C++ has the same name as class or structure. There can be two types of constructors in C++.

1. Default constructor : Aconstructor which has no argument is known as default constructor. It is invoked at the time of creating an object.

2. Parameterized constructor : Constructor which has parameters is called a parameterized constructor. It is used to provide different values to distinct objects.

Example :

```
class Hero{
    public:
    int health;
        Hero(){
            cout<<"constructor called"<<endl;
        }
        //paramerterised constrcutor
        Hero(int health){
            cout<<"this->"<<this<<endl;
            this->health=health;
        }
};
```

Q4. Write down the use of destructor in C++.

● Destructor : A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically. Adestructor is defined like a constructor. It must have the same name as class, prefixed with a tilde sign (~).

Q5. What is the need of constructor? How it is different from the member function?

● Constructor : Constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally. The constructor in C++ has the same name as class or structure.

How it is different from the member function?

1) Constructor doesn't have a return type. Member function has a return type.
2) Constructor is automatically called when we create the object of the class. Member function needs to be called explicitly using object of class.
3) When we do not create any constructor in our class, C++ compiler generates a default constructor and insert it into our code. The same does not apply to member functions

Q6. What is a static data member? How they are used in static functions? Explain with suitable illustrations.

Static data members are class members that are declared using static keywords. A static member has certain special characteristics. These are:

- Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
- It is initialized before any object of this class is being created, even before main starts.
- It is visible only within the class, but its lifetime is the entire program

```
#include <bits/stdc++.h>
using namespace std;
class Hero{
    public:
        int age;
        string name;
        static int count;
        static int random(){
            return count;//only take static objects
        }
};
int Hero::count=56;
int main() {
    cout<<Hero::count<<endl;
    cout<<Hero::random()<<endl;
    return 0;
}
```

Q7. Define class and objects.

● Class is a user-defined data type which defines its properties and its functions. Class is the only logical representation of the data. For example, Human being is a class. The body parts of a human being are its properties, and the actions performed by the body parts are known as functions. The class does not occupy any memory space till the time an object is instantiated.

● Object is a run-time entity. It is an instance of the class. An object can represent a person, place or any other item. An object can operate on both data members and member functions.

Q8. What do you mean by dynamic binding? How it is useful in OOP?

Dynamic binding

The general meaning of binding is linking something to a thing. Here linking of objects is done. In a programming sense, we can describe binding as linking function definition with the function call.

So the term dynamic binding means to select a particular function to run until the runtime. Based on the type of object, the respective function will be called.

As dynamic binding provides flexibility, it avoids the problem of static binding as it happened at compile time and thus linked the function call with the function definition.

Use of dynamic binding

dynamic binding also helps us to handle different objects using a single function name. It also reduces the complexity and helps the developer to debug the code and error.

Q9. Explain the use of friend function with the help of suitable example.

● Friend Function : Friend function acts as a friend of the class. It can access the private and protected members of the class. The friend function is not APNI KAKSHA a member of the class, but it must be listed in the class definition. The non-member function cannot access the private data of the class. Sometimes, it is necessary for the non-member function to access the data. The friend function is a non-member function and has the ability to access the private data of the class.

Note :

1. Afriend function cannot access the private members directly, it has to use an object name and dot operator with each member name.

2. Friend function uses objects as arguments.

```
class Node {
private:
    int key;
    Node* next;
    friend class LinkedList;
};
```

Q10. What is the need of overloading operators and functions?

Function overloading reduces the investment of different function names and used to perform similar functionality by more than one function.

overloaded operators provides syntactic sugar for function calls that are equivalent. Without adding to / changing the fundamental language changes, operator overloading provides a pleasant façade

Q11. How do we invoke constructor? Can we have more than one constructor in a class? If yes, explain the need for such a situation.

yes, we can have more than one constructor in a class

```
class Hero{
    public:
    int health;
        Hero(){
            cout<<"constructor called"<<endl;
        }
        Hero(int health){
            cout<<"this->"<<this<<endl;
            this->health=health;
        }
};
```

A class can have multiple constructors that assign the fields in different ways. Sometimes it's beneficial to specify every aspect of an object's data by assigning parameters to the fields, but other times it might be appropriate to define only one or a few

Q12. Write down the example to overload unary and binary operators in C++.

overload unary

```
#include <iostream>
using namespace std;
class Distance {
public:
    int feet, inch;
    Distance(int f, int i){
        this->feet = f;
        this->inch = i;
    }
    void operator-(){
        feet--;
        inch--;
        cout << "\nFeet & Inches(Decrement): " << feet <<
"'" << inch;
    }
};
int main(){
    Distance d1(8, 9);
    -d1;
    return 0;
}
```

overload binary

```
#include <iostream>
using namespace std;
class Distance {
public:
    int feet, inch;
    Distance(){
        this->feet = 0;
        this->inch = 0;
    }
    Distance(int f, int i){
        this->feet = f;
```

```
        this->inch = i;
    }
    Distance operator+(Distance& d2){
        Distance d3;
        d3.feet = this->feet + d2.feet;
        d3.inch = this->inch + d2.inch;
        return d3;
    }
};
int main(){
    Distance d1(8, 9);
    Distance d2(10, 2);
    Distance d3;
    d3 = d1 + d2;
    cout << "\nTotal Feet & Inches: " << d3.feet << "'" <<
d3.inch;
    return 0;
}
```

Q13. State the use of scope resolution operator in C++.

The scope resolution operator is used to reference the global variable or member function that is out of scope. Therefore, we use the scope resolution operator to access the hidden variable or function of a program. The operator is represented as the double colon (::) symbol.

Uses of the scope resolution Operator

1. It is used to access the hidden variables or member functions of a program.
2. It defines the member function outside of the class using the scope resolution.
3. It is used to access the static variable and static function of a class.
4. The scope resolution operator is used to override function in the Inheritance.

Q14. Compare and contrast the structured programming and object oriented programming.

Structured Programming	Object-Oriented Programming
It is a subset of procedural programming.	It relies on concept of objects that contain data and code.
Programs are divided into small programs or functions.	Programs are divided into objects or entities.
It is all about facilitating creation of programs with readable code and reusable components.	It is all about creating objects that usually contain both functions and data.
Its main aim is to improve and increase quality, clarity, and development time of computer program.	Its main aim is to improve and increase both quality and productivity of system analysis and design.
It simply focuses on functions and processes that usually work on data.	It simply focuses on representing both structure and behavior of information system into tiny or small modules that generally combines data and process both.
It is a method of organizing, managing and coding programs that can give or provide much easier modification and understanding.	It is a method in which set of objects can vary dynamically and can execute just by acting and reading to each other.
In this, methods are written globally and code lines are processed one by one i.e., Run sequentially.	In this, method works dynamically, make calls as per need of code for certain time.
It generally follows “Top-Down Approach”.	It generally follows “Bottom-Up Approach”.
It provides less flexibility and abstraction as compared to object-oriented programming.	It provides more flexibility and abstraction as compared to structured programming.

It is more difficult to modify structured program and reuse code as compared to object-oriented programs.	It is less difficult to modify object-oriented programs and reuse code as compared to structured programs.
It gives more importance of code.	It gives more importance to data.
Example : Pascal, ALGOL, C, Modula-2, etc.	Example : JAVA, C#, C++, etc.

Q15. What is a dynamic constructor? Explain with suitable example.

When allocation of memory is done dynamically using dynamic memory allocator new in a constructor, it is known as dynamic constructor. By using this, we can dynamically initialize the objects

exmaple:

```
#include <iostream>
using namespace std;
class geeks {
    const char* p;
public:
    geeks(){
        p = new char[6];

    }
    void display(){
        cout << p << endl;
    }
};
int main(){
    geeks obj;
    obj.display();
}
```

Q16. Define a structure that represents Fruit with properties fruit name, fruit type, fruit color. Write a program that accepts data of four fruits and displays the results.

note: I think this is not in syllabus or might come as class rather then struct

```
#include <iostream>
using namespace std;
struct Fruit{
  char name[50], type[50], color[50];
  int roll;
} f[4];
int main(){
  cout << "Enter Fruit and fruit properties: " << endl;
  for(int i = 0; i < 4; ++i){
    f[i].roll = i+1;
    cout << "For roll number" << f[i].roll << "," << endl;
    cout << "Enter fruit name: ";
    cin >> f[i].name;
    cout << "Enter fruit type: ";
    cin >> f[i].type;
    cout << "Enter fruit color: ";
    cin >> f[i].color;
    cout << endl;
  }
  cout << "Displays the results: " << endl;
  for(int i = 0; i < 4; ++i){
    cout << "\nRoll number: " << i+1 << endl;
    cout << "Fruit Name: " << f[i].name << endl;
    cout << "Fruit type: " << f[i].type << endl;
    cout << "Fruit color: " << f[i].color << endl;
  }
  return 0;
}
```

Q17. Compare and Contrast late binding and early binding.

Early Binding (compile-time time polymorphism) As the name indicates, compiler (or linker) directly associate an address to the function call. It replaces the call with a machine language instruction that tells the mainframe to leap to the address of the function.

Late Binding : (Run time polymorphism) In this, the compiler adds code that identifies the kind of object at runtime then matches the call with the right function definition (Refer this for details). This can be achieved by declaring a virtual function.

Early Binding

```
#include<iostream>
using namespace std;
class Base{
public:
    void show() { cout<<" In Base \n"; }
};
class Derived: public Base{
public:
    void show() { cout<<"In Derived \n"; }
};
int main(void){
    Base *bp = new Derived;
    bp->show();
    return 0;
}
```

Late Binding

```
#include<iostream>
using namespace std;
class Base{
public:
    virtual void show() { cout<<" In Base \n"; }
};
class Derived: public Base{
public:
    void show() { cout<<"In Derived \n"; }
};
int main(void){
    Base *bp = new Derived;
    bp->show();
    return 0;
}
```

Q18. What do you mean by implicit and explicit call of constructor? Explain with example.

implicit	explicit

class MyClass {
  int i;
  /*implicit*/ MyClass(int i) : i(i) {}
}
 	
// ...
 	
int main() {
  MyClass clz = 2;
}
	class MyClass {
	  int i;
	  explicit MyClass(int i) : i(i) {}
	};
	
	int main() {
	  MyClass clz = MyClass(2);
	}

These constructors allow you to initialize a class value without specifying the name of the class	You must initialize the value with the name of the type before it's initialization aka: MyClass then with the constructor parameters. You can see the difference in usability in the implicit constructor code

Explicit means done by the programmer. Implicit means done by the JVM or the tool , not the Programmer.

Q19. Write a C++ program to overload area() function to calculate area of shapes like triangle ,square, circle.

```
#include<bits/stdc++.h>
using namespace std;
class Circle{
    public:
    void area(int radius){
        int areaa=3.14*radius*radius;
        cout<<"the area of circle is: "<<areaa<<endl;
    }
};
class Triangle{
    public:
    void area(int base,int height){
      int areaa=0.5*base*height;
      cout<<"area of trianle is :"<<areaa<<endl;
    }

};
class Square{
    public:
    void area(int side){
        int areaa=side*side;
        cout<<"area of square is :"<<areaa<<endl;
    }
};
int main(){
    Circle c;
    Triangle t;
    Square s;
    c.area(4);
    t.area(4,6);
    s.area(7);
    return 0;
}
```